

UTILITY PATENT APPLICATION TRANSMITTAL

Submit an original and a duplicate for fee processing
(Only for new nonprovisional applications under 37 CFR §1.53(b))

ADDRESS TO:

Commissioner of Patents and Trademarks
Box Patent Application
Washington, D.C. 20231

Attorney Docket No. 203993

First Named Inventor Crites

Express Mail No. EL643535392US

APPLICATION ELEMENTS

1. ☒ Utility Transmittal Form
2. ☒ Specification (including claims and abstract) [Total Pages 26]
3. ☒ Drawings [Total Sheets 3]
4. ☐ Combined Declaration and Power of Attorney [Total Pages]
 - a. ☐ Newly executed
 - b. ☐ Copy from prior application
[Note Box 5 below]
 - i. ☐ Deletion of Inventor(s) Signed statement attached deleting inventor(s) named in the prior application
5. ☐ Incorporation by Reference: The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Microfiche Computer Program
7. ☐ Nucleotide and/or Amino Acid Sequence Submission
 - a. ☐ Computer Readable Copy
 - b. ☐ Paper Copy
 - c. ☐ Statement verifying above copies

ACCOMPANYING APPLICATION PARTS

8. ☐ Assignment Papers (cover sheet and document(s))
9. ☐ Power of Attorney
10. ☐ English Translation Document (if applicable)
11. ☐ Information Disclosure Statement (IDS)
 - ☐ Form PTO-1449
 - ☐ Copies of References
12. ☐ Preliminary Amendment
13. ☒ Return Receipt Postcard (Should be specifically itemized)
14. ☐ Small Entity Statement(s)
 - ☐ Enclosed
 - ☐ Statement filed in prior application; status still proper and desired
15. ☐ Certified Copy of Priority Document(s)
16. ☒ Other: Appendix 1

17. If a **CONTINUING APPLICATION**, check appropriate box and supply the requisite information in (a) and (b) below:

- (a) ☐ Continuation ☐ Divisional ☐ Continuation-in-part of prior application Serial No. _____
Prior application information: Examiner _____; Group Art Unit: _____
- (b) Preliminary Amendment: Relate Back - 35 USC §120. The Commissioner is requested to amend the specification by inserting the following sentence before the first line:
"This is a ☐ continuation ☐ divisional of copending application(s)
☐ Application No. _____, filed on _____,
☐ International Application, filed on _____, and which designates the U.S."

APPLICATION FEES


BASIC FEE				\$690.00
CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	
Total Claims	42 -20=	22	x \$18.00	\$396.00
Independent Claims	4 - 3=	1	x \$78.00	\$78.00
<input type="checkbox"/> Multiple Dependent Claims(s) if applicable			+ \$260.00	\$
Total of above calculations =				\$1164.00
Reduction by 50% for filing by small entity =				\$()
<input type="checkbox"/> Assignment fee if applicable			+ \$40.00	\$
TOTAL =				\$1164.00

UTILITY PATENT APPLICATION TRANSMITTAL

Attorney Docket No. 203993

19. ☒ Please charge my Deposit Account No. 12-1216 in the amount of \$1164.00.
20. ☐ A check in the amount of \$ is enclosed.
21. The Commissioner is hereby authorized to credit overpayments or charge any additional fees of the following types to Deposit Account No. 12-1216:
- a. ☒ Fees required under 37 CFR §1.16.
- b. ☒ Fees required under 37 CFR §1.17.
22. ☒ The Commissioner is hereby generally authorized under 37 CFR §1.136(a)(3) to treat any future reply in this or any related application filed pursuant to 37 CFR §1.53 requiring an extension of time as incorporating a request therefor, and the Commissioner is hereby specifically authorized to charge Deposit Account No. 12-1216 for any fee that may be due in connection with such a request for an extension of time.

23. CORRESPONDENCE ADDRESS

<input checked="" type="checkbox"/> Customer Number: 23460  23460 PATENT TRADEMARK OFFICE		<input type="checkbox"/> Reg. No. Leydig, Voit & Mayer, Ltd. Two Prudential Plaza, Suite 4900 180 North Stetson Chicago, Illinois 60601-6780 (312) 616-5600 (telephone) (312) 616-5700 (facsimile)
Name	Kevin L. Wingate, Registration No. 38,662	
Signature	<i>Kevin L. Wingate</i>	
Date	July 31, 2000	

Certificate of Mailing Under 37 CFR §1.10

I hereby certify that this Utility Patent Application Transmittal and all accompanying documents are being deposited with the United States Postal Service "Express Mail Post Office To Addressee" Service under 37 CFR §1.10 on the date indicated below and is addressed to: Commissioner of Patents and Trademarks, Box Patent Application, Washington, D.C. 20231.

Matthew W. Olson	<i>Matthew W. Olson</i>	July 31, 2000
Name of Person Signing	Signature	Date

UTILITY (Rev. 6/29/2000)

A FLEXIBLE INTERFACE FOR CONTROLLING STREAMING DATA IN PASSIVE STREAMING PLUG-INS

5

TECHNICAL FIELD

This invention relates generally to electronic data processing, and, more particularly, to managing the flow of streaming data through a processing module in a computer system.

10

COPYRIGHT NOTICE/PERMISSION

15

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever. The following notice applies to the software and data as described below and in the drawings hereto: Copyright © 2000, Microsoft Corporation, All Rights Reserved.

20

BACKGROUND OF THE INVENTION

25

Continued advances in computer technology have lead to an explosion in the use of multimedia applications. Computer based multimedia, the combination of video and audio in a digital format for viewing on a computer, involves a multitude of electronic circuits and subsystems. It is estimated that more than 100 million computers are now equipped with multimedia hardware and software. Nearly every new personal computer manufactured today includes some form of multimedia. Furthermore, digital products

having some form of multimedia, such as cameras and video recorders, are now hitting the market in a dizzying succession. Multimedia has not only turned the computer into an indispensable information, education, and entertainment tool, but it has also revolutionized the way society works, entertains, and stays informed. Multimedia has also helped drive the computer industry to provide tools that can be used by novice computer users making computers nearly as prevalent as radios and televisions.

Multimedia is also becoming increasingly prevalent in the Internet realm as the growth of the Internet steadily and rapidly continues. A technique known as streaming has been developed for multimedia applications that allows data to be transferred so that it can be processed as a steady and continuous stream. This has the benefit that data can be displayed before the entire file has been transmitted, a must for large multimedia files.

Streaming technologies are becoming increasingly important with the growth of the Internet because most users do not have fast enough Internet access to download large multimedia files quickly.

Streaming data almost always requires some form of processing among various modules or filters in a system. Compression and decompression of audio and video data as well as the use of software to create special effects on that data are typical of the types of processing applied to streaming data. For example, a video clip might require MPEG decoding in a dedicated hardware module, rasterizing the video fields in another hardware module, digital filtering of the audio in a software module, insertion of subtitles by another software module, parsing audio data to skip silent periods by a software module, D/A conversion of the video in a video adapter card, and D/A conversion of the audio in a separate audio card.

As these technologies were developed to process streaming data, the concept of a graph was introduced for specifying the connections among the modules through which a data stream must pass in an effort to increase the data processing speed. Figure 4 is representative of the graph concept. In these graphs, an application 36 communicates with a graph manager 400 to indicate what the application wants done. The graph manager selects which filter modules 402, 404, 406 to use and the modules 402, 404, 406 within the graph negotiate directly with each other. Protocols have been developed to specify the flow of data through a graph, and to specify the control protocol that adjacent modules in the graph use to communicate with each other to request and accept the data. During connection of modules in a graph, these protocols define a predefined fixed sequence of data flow 408 and control connection negotiations 410 in a graph. A typical negotiation sequence negotiates the following in order: the interface, the medium, the data format, the allocators, and the master clock.

These implementations have several limitations. One limitation in these systems is that an application is forced to use a graph manager to select which modules to use and is not allowed to select the medium, the format to use, the allocators, the threads, etc. Another limitation is that the modules used for encoding and decoding use essentially different streaming processes and have different interfaces for audio and video and for compression and decompression.

Accordingly, there exists a continued need for further efficiencies in processing streaming and related types of data by providing control mechanisms that achieve the efficiency of a dedicated protocol while allowing enough flexibility to use different data

types, different modules, and different configurations in the environment of streaming data through multiple processing modules.

5

SUMMARY OF THE INVENTION

In view of the above described problems existing in the art, the present invention provides a flexible interface that enables an application to communicate directly with processing modules and easily control the processing of streaming data. The interface provides basic commands that allow applications to communicate with processing
10 modules and adapt to changing standards.

The interface enables an application to set the input-data format of the input to a processing module and set the output-data format of the output of the processing module.

Once the input-data and output-data formats are set, the application uses the interface to control when the processing module both processes input data and generates output data.

15 The processing module sets a flag to signal the application when the module is unable to generate all the output data for the associated input data.

Applications know the capabilities of processing modules by having the modules enumerate their capabilities via the interface. A processing module enumerates its capabilities by category, by media type, or by both category and media type. Processing
20 modules use the interface to register themselves and are registered by class ID, category, whether the application needs a key, the number and types of input data types, and the number and type of output data types to register.

Additional features and advantages of the invention will be made apparent from the following detailed description of illustrative embodiments which proceeds with reference to the accompanying figures.

BRIEF DESCRIPTION OF THE DRAWINGS

While the appended claims set forth the features of the present invention with particularity, the invention, together with its objects and advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

Figure 1 is a block diagram generally illustrating an exemplary computer system on which the present invention resides;

Figure 2 is a block diagram of an application controlling objects in accordance with the teaching of the instant invention;

Figure 3 is a flowchart of a method to control an object in accordance with the teaching of the instant invention; and

Figure 4 shows a prior art graph control concept.

DETAILED DESCRIPTION OF THE INVENTION

Turning to the drawings, wherein like reference numerals refer to like elements, the invention is illustrated as being implemented in a suitable computing environment. Although not required, the invention will be described in the general context of computer-executable instructions, such as program modules, being executed by a personal computer. Generally, program modules include routines, programs, objects, components,

data structures, etc. that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multi-processor systems, microprocessor based or programmable consumer electronics, network PCs, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

With reference to Fig. 1, an exemplary system for implementing the invention includes a general purpose computing device in the form of a conventional personal computer 20, including a processing unit 21, a system memory 22, and a system bus 23 that couples various system components including the system memory to the processing unit 21. The system bus 23 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. The system memory includes read only memory (ROM) 24 and random access memory (RAM) 25. A basic input/output system (BIOS) 26, containing the basic routines that help to transfer information between elements within the personal computer 20, such as during start-up, is stored in ROM 24. The personal computer 20 further includes a hard disk drive 27 for reading from and writing to a hard disk, not shown, a magnetic disk drive 28 for reading from or writing to a removable magnetic disk 29, and an optical disk drive 30 for reading from or writing to a removable optical disk 31 such as a CD ROM or other optical media.

The hard disk drive 27, magnetic disk drive 28, and optical disk drive 30 are connected to the system bus 23 by a hard disk drive interface 32, a magnetic disk drive interface 33, and an optical disk drive interface 34, respectively. The drives and their associated computer-readable media provide nonvolatile storage of computer readable instructions, data structures, program modules and other data for the personal computer 20. Although the exemplary environment described herein employs a hard disk, a removable magnetic disk 29, and a removable optical disk 31, it will be appreciated by those skilled in the art that other types of computer readable media which can store data that is accessible by a computer, such as magnetic cassettes, flash memory cards, digital video disks, Bernoulli cartridges, random access memories, read only memories, and the like may also be used in the exemplary operating environment.

A number of program modules may be stored on the hard disk, magnetic disk 29, optical disk 31, ROM 24 or RAM 25, including an operating system 35, one or more applications programs 36, other program modules 37, and program data 38. A user may enter commands and information into the personal computer 20 through input devices such as a keyboard 40 and a pointing device 42. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 21 through a serial port interface 46 that is coupled to the system bus, but may be connected by other interfaces, such as a parallel port, game port or a universal serial bus (USB). A monitor 47 or other type of display device is also connected to the system bus 23 via an interface, such as a video adapter 48. In addition to the monitor, personal computers typically include other peripheral output devices, not shown, such as speakers and printers.

The personal computer 20 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 49. The remote computer 49 may be another personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the personal computer 20, although only a memory storage device 50 has been illustrated in Fig. 1. The logical connections depicted in Fig. 1 include a local area network (LAN) 51 and a wide area network (WAN) 52. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

When used in a LAN networking environment, the personal computer 20 is connected to the local network 51 through a network interface or adapter 53. When used in a WAN networking environment, the person computer 20 typically includes a modem 54 or other means for establishing communications over the WAN 52. The modem 54, which may be internal or external, is connected to the system bus 23 via the serial port interface 46. In a networked environment, program modules depicted relative to the personal computer 20, or portions thereof, may be stored in the remote memory storage device. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

In the description that follows, the invention will be described with reference to acts and symbolic representations of operations that are performed by one or more computer, unless indicated otherwise. As such, it will be understood that such acts and operations, which are at times referred to as being computer-executed, include the manipulation by the processing unit of the computer of electrical signals representing data

in a structured form. This manipulation transforms the data or maintains it at locations in the memory system of the computer, which reconfigures or otherwise alters the operation of the computer in a manner well understood by those skilled in the art. The data structures where data is maintained are physical locations of the memory that have particular properties defined by the format of the data. However, while the invention is being described in the foregoing context, it is not meant to be limiting as those of skill in the art will appreciate that various of the acts and operation described hereinafter may also be implemented in hardware.

Figure 2 shows a representative block diagram of data flow from the application 36 to objects 100, 102, 104 in the operating system 35. Application 36 is in communication with objects 100, 102, 104 via control lines 106, 108, 110. Application 36 represents any application that uses objects to manipulate data. For example, application 36 could be a computer game, a media player such as Windows Media™, etc. An object can be designed to perform any type of data processing. For example, an object could be an encoder, a decoder, a reader, a renderer, an audio effect processor, a video effect processor, etc. The application 36 sends data to the objects on input data connections 112, 114, 116 via input buffers 124, 126, 128 and receives data from objects on output data connections 118, 120, 122 via output buffers 125, 127, 129. The input buffers 124, 126, 128 and output buffers 125, 127, 129 are located in RAM 25 (see figure 1). While figure 2 only shows objects having one input and output data connection, those skilled in the art will recognize that each object may have any number of input and output data connections.

The invention will be described in the context of the Windows operating system from Microsoft Corporation. However, it should be understood that the instant invention is applicable to other operating systems. Figure 3 shows a flow chart of the steps taken to process data in accordance with the instant invention. For the following description,

5 application 36 shall be communicating with object 100, although it should be recognized that in general, application 36 communicates with all objects. Exemplary commands of one embodiment shall be used to illustrate the instant invention. See appendix 1 for the specific details of the exemplary command. Appendix 1 provides a listing of commands for one specific implementation of the invention. It should be noted that the exemplary
10 command is not limited to the specific implementation of this embodiment and the invention is not limited to the commands of appendix 1 or to the Windows operating system. Those skilled in the art will recognize that the functions of the exemplary command can be implemented in other operating systems.

As standards change and new media types are created, the instant invention
15 provides the application 36 the capability to determine the media types (i.e., data format) that an input of an object can accept using the GetInputType command and the media types that an object can output using the GetOutputType command. The minimum size of an object's input and output buffer sizes can be found using the GetInputSizeInfo and GetOutputSizeInfo commands. Using the minimum size of input and output buffers
20 guarantees that some data is processed. These sizes should be determined after the media type has been set because a different buffer size may be required for different media types.

The application 36 sets the media type of the input stream of object 100 using the SetInputType command (step 132) and the output media type of the output stream of object 100 using the SetOutputType command (step 134). The input media type and the output media type can be set before data is processed or while data is being processed.

- 5 They are set in the context of each other and in the context of whether data is being streamed. The object rejects an input media type or an output media type if the media type is inconsistent with other media types or the resources allocated. In general, the media type of input streams should be set before the media types of output streams. Some objects may not enumerate media types for its output until after the input stream
- 10 media type is set. For example, a decoder with one input stream and one output stream might only enumerate media types for its output when the media type for its input has been set. Once the media types are set, the application 36 commands the object 100 to process data in an input buffer using the Process Input command (step 136). The application 36 may want to command the object to process the input data before setting
- 15 the output media type. The instant invention provides the application 36 flexibility by not requiring the application 36 to set the output media type before commanding the object 100 to process input data. The application 36 merely commands the object 100 to process input data before setting the output media type. Upon receiving the command to process input data, the object 100 either processes all the data in the buffer or holds the
- 20 buffer waiting until it is commanded to generate output data. When the object 100 holds the buffer, the application 36 is notified so that it does not reuse the buffer.

The object 100 may want to hold on to multiple input buffers. One reason for this to occur is for the object 100 to perform a lookahead function. The application 36 knows

when the object 100 may hold on to multiple input buffers by detecting a flag set by the object when the application gets information about the input stream using the `GetInputStreamInfo` command. The application 36 can determine the minimum size of an object's input buffer size required to guarantee that some data is processed using the `GetInputSizeInfo` command. The application allocates a sufficient number of buffers for the object to avoid running out of buffers, the number defined by

$$number \geq \frac{(\min imumdatasize + 2 * (buffersize - 1))}{buffersize}$$

where *minimumdatasize* is the minimum required size of an object's input buffer and *buffersize* is the size of the buffers allocated by the application.

There are instances when the input data stream is discontinuous. For example, this could occur when there is a large gap in the data, when no more data is expected, or when the format of the data changes. Some objects may need to output extra data when there is no more input data or when there is a logical break in the input data. For example, some audio encoder objects generate partial data samples, some encoder objects perform lookahead, and some decoder objects defer reference frames so that the first decoded frame generates no output and the last decoded frame forces out the last reference frame. The application 36 detects when the input stream data is discontinuous (step 138). If the input stream data is discontinuous, the application 36 informs the object 100 that the input stream data is discontinuous using the `Discontinuity` command (step 140). The object 100 generally should generate all output data that can be generated from the data already received before accepting any more input data when the application commands the object to generate data using the `Process Output` command (step 142). If the input stream data is continuous, the application 36 commands the object 100 to

process the data to generate output data using the ProcessOutput command (step 144). In some situations, output data may not be generated for input data. For example, an output buffer 125 may not be filled if the media type being used requires complete samples to be generated and not enough input data has been received to generate a complete output buffer 125. In one embodiment, object 100 provides a status to indicate to the application 36 that there is no output data to process.

The object 100 sets a flag in the output buffer 125 associated with an output data stream to signal the application 36 that more buffers are needed to generate the output. The flag allows the application 36 to avoid allocating an output buffer before it is needed. For example, the flag may be set because an output buffer 125 may not be large enough for all the output data, or the object 100 may output data in a number of portions due to the way a particular media type is defined to flow, or the object 100 needs to output a timestamp with the next batch of data. When the application 36 detects this flag, the application 36 continues to command the object 100 to generate output data until the flag is no longer present. If the input data to the object is timestamped, the object timestamps the output data as the object generates output data. Once the output data is generated, the application 36 can continue to process data by repeating the commands to process input data and output data.

New objects are registered in the operating system 35 using the DMORRegister command. In the exemplary embodiment, the objects are registered in the system registry. The object registers the class ID under which the object is registered, the category of the object, whether a key is needed to use the object, and the number and data

types of input and outputs of the object. The objects can be enumerated by category, by media type, or by category and media type using the DMOEnum command.

An interface that enables an application to directly control streaming data processing and to directly communicate with processing modules has been described.

- 5 Applications set the input and output media types of an object, directly control when an object processes input data or generates output data, and knows the media types an object supports by having objects enumerate their capabilities.

In view of the many possible embodiments to which the principles of this invention may be applied, it should be recognized that the embodiment described herein

- 10 with respect to the drawing figures is meant to be illustrative only and should not be taken as limiting the scope of invention. For example, those of skill in the art will recognize that the elements of the illustrated embodiment shown in software may be implemented in hardware and vice versa or that the illustrated embodiment can be modified in arrangement and detail without departing from the spirit of the invention.

- 15 Therefore, the invention as described herein contemplates all such embodiments as may come within the scope of the following claims and equivalents thereof.

CLAIMS**We claim:**

1. A computer-readable medium having computer-executable instructions for performing the steps of:

- 5 commanding the object to set a first data type on an input of the object;
commanding the object to set a second data type on an output of the object;
commanding the object to process data of the first type received at the input of the object; and
commanding the object to generate output data of the second type at the output of
10 the object.

2. The computer-readable medium of claim 1 having further computer-executable instructions for performing the steps of:

- detecting when an incomplete status flag is set; and
15 re-commanding the object to generate output data on the output of the object if the incomplete status flag is set.

3. The computer-readable medium of claim 2 wherein the step of re-commanding the object to generate output data on the output of the object further comprises repeating
20 the step of re-commanding the object to generate output data for as long as the incomplete status flag is set.

4. The computer-readable medium of claim 1 wherein the step of commanding the object to set the first data type comprises the step of commanding the object to set the first data type to a streaming media input type and the step of commanding the object to set the second data type comprises the step of commanding the object to set the second data type to a streaming media output type.

5. The computer-readable medium of claim 4 wherein the step of commanding the object to set the first data type to a streaming media input type comprises the step of commanding the object to set the streaming media input type to a streaming audio input media type and the step of commanding the object to set the second data type to a streaming media output type comprises the step of commanding the object to set the streaming media output type to a streaming audio output media type.

6. The computer-readable medium of claim 4 wherein the step of commanding the object to set the first data type to a streaming media input type comprises the step of commanding the object to set the streaming media input type to a streaming video input media type and the step of commanding the object to set the second data type to a streaming media output type comprises the step of commanding the object to set the streaming media output type to a streaming video output media type.

7. The computer-readable medium of claim 1 having further computer-executable instructions for performing the steps of:

detecting the presence of an incomplete status flag; and

if the incomplete status flag is set, waiting for the incomplete status flag to reset before performing the step of commanding the object to process data of the first type received at the input of the object.

5 8. The computer-readable medium of claim 1 having further computer-executable instructions for performing the step of querying the object for a minimum input buffer size required to guarantee that some data is processed when the object is commanded to process data of the first type.

10 9. The computer-readable medium of claim 1 having further computer-executable instructions for performing the step of querying the object for a minimum output buffer size required to guarantee that some data is output when the object is commanded to generate output data .

15 10. The computer-readable medium of claim 1 having further computer-executable instructions for performing the steps of:

determining an first data type that the object can process on the input; and

determining a second data type that the object can support on the output.

20 11. The computer-readable medium of claim 1 having further computer-executable instructions for performing the step of informing the object that the data is discontinuous on the input of the object.

12. The computer-readable medium of claim 1 having further computer-executable instructions for performing the steps of:

determining a first data type that the object can process on the input;

determining a second data type that the object can support on the output;

5 querying the object for a minimum input buffer size required to guarantee that

some data is processed when the object is commanded to process input data;

detecting when an incomplete status flag is set;

re-commanding the object to generate output data on the output of the object if the incomplete status flag is set; and

10 if the incomplete status flag is set, waiting for the incomplete status flag to reset before performing the step of commanding the object to process data on the input of the object.

13. The computer-readable medium of claim 12 having further computer-executable
15 instructions for performing the step of informing the object that data is discontinuous on the input of the object.

14. The computer-readable medium of claim 1 wherein the step of commanding the object to generate output data on the output of the object further comprises the step of
20 detecting that an output flag has been set to indicate that the output data can be generated prior to commanding the object to generate output data on the output of the object.

15. A computer-readable medium having computer-executable instructions for performing the steps of:

setting input and output data types for a respective input and output of an object in response to at least one command from an application;

5 processing input data on the input of the object in response to a command from the application to process data on the input; and

generating output data on the output of the object in response to a command from the application to generate data on the output.

10 16. The computer-readable medium of claim 15 wherein the step of setting the input and output data types comprises the step of setting the input data type to a streaming media input type and of setting the output data type to a streaming media output type.

15 17. The computer-readable medium of claim 16 wherein the step of setting the streaming media input type comprises the step of setting the streaming media input type to a streaming audio input media type and the step of setting the streaming media output type comprises the step of setting the streaming media output type to a streaming audio output media type.

20 18. The computer-readable medium of claim 16 wherein the step of setting the streaming media input type comprises the step of setting the streaming media input type to an input streaming video media type and the step of setting the streaming media output

type comprises the step of setting the streaming media output type to an output streaming video media type.

19. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of setting an incomplete status flag upon receiving a command from the application to generate output data if all output data for the associated input data cannot be generated.

20. The computer-readable medium of claim 19 having further computer-executable instructions for performing the step of resetting the incomplete status flag upon generating all output data for the associated input data.

21. The computer-readable medium of claim 19 having further computer-executable instructions for performing the steps of buffering input data internally when there is insufficient input data to generate output data.

22. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of providing an indication that the output data can be generated.

23. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of enumerating types of data that are supported in response to a query from the application.

24. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of generating all data that can be processed in response to notice from the application that data is discontinuous on the input.

25. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of setting a buffer flag in response to a query to provide information about an input data stream, the buffer flag indicating that a plurality of input buffers may be held.

26. The computer-readable medium of claim 25 having further computer-executable instructions for performing the steps of:

setting a lookahead value, the lookahead value indicating a maximum size of data held by the object; and
providing the lookahead value to the application in response to a command from the application to provide buffer size requirements for the input data stream.

27. The computer-readable medium of claim 26 wherein the application has a fixed buffer size, the computer-readable medium having further computer-executable instructions for performing the the step of allocating a number of buffers for processing data, defined by

$$number \geq \frac{(\max imumdatasize + 2 * (fixedbuffersize - 1))}{fixedbuffersize}$$

28. The computer-readable medium of claim 15 having further computer-executable instructions for performing the step of registering an existence with an operating system.

5 29. The computer-readable medium of claim 28 wherein the step of registering an existence with an operating system comprises:

identifying a class ID;

identifying a category;

identifying whether a use is keyed;

10 identifying a number of input data types to register;

identifying the input data types;

identifying a number of output data types to register; and

identifying the output data types.

15 30. A method of configuring and controlling an object for processing data, the method comprising the steps of:

issuing from a process one or more commands to set a data type at each of a data input and a data output of the object;

20 setting the data type for each of the data input and data output of the object in response to the command;

processing data presented to the input of the object in response to a command from the application to begin processing data; and

generating output data derived from the input data in response to a command from the application.

31. The method of claim 31 further comprising the steps of:

5 issuing an incomplete status flag if the object is unable to generate all the output data for associated input data; and

reissuing from the process a command to generate output data upon receiving the incomplete status flag.

10 32. The method of claim 31 further comprising the step of issuing a reset status flag upon generating all output data for the associated input data.

33. The method of claim 30 further comprising the step of issuing an indication that the object can generate output data.

15 34. The method of claim 30 further comprising the steps of:

issuing from the process a data type query for the types of data the object supports; and

enumerating the types of data supported in response to the data type query.

20 35. The method of claim 30 further comprising the steps of:

issuing from the process an input discontinuity notice; and

36. The method of claim 30 further comprising the steps of:

issuing from the process an information query to provide information about an input stream; and

setting a buffer flag indicating that a plurality of input buffers may be held in response to the information query.

10 37. The method of claim 36 further comprising the steps of:

issuing from the process a buffer size requirement command; and

setting and issuing a lookahead value in response to the buffer size requirement

command, the lookahead value indicating a maximum size of data held by the object.

38. An interface for enabling applications to control modules for processing streaming media data, the interface comprising: a first command to set an input data format of the processing module; a second command to set an output data format of the processing module; a third command to process data on an input of the processing module; and a fourth command to generate data on an output of the processing module.

39. The interface of claim 38 further comprising a fifth command to enumerate the capabilities of a processing module by at least one of a category and a media type.

40. The interface of claim 38 further comprising: a sixth command to determine a minimum input buffer size of a processing module required to guarantee that data is processed; and a seventh command to determine a minimum output buffer size of a processing module required to guarantee that data is generated.

5

41. The interface of claim 38 further comprising: an eighth command to determine the input data types that a processing module can process; and a ninth command to determine the output data types that a processing module can generate.

10

42. The interface of claim 38 further comprising: a fifth command to enumerate the capabilities of a processing module by at least one of a category and a media type; a sixth command to determine a minimum input buffer size of a processing module required to guarantee that data is processed; a seventh command to determine a minimum output buffer size of a processing module required to guarantee that data is generated; an eighth

15

command to determine the input data types that a processing module can process; and a ninth command to determine the output data types that a processing module can generate.

DRAFT: 5/26/2016

ABSTRACT OF THE INVENTION

A flexible interface that enables an application to communicate directly with processing modules to easily control the processing of streaming data. The interface provides basic commands for applications to use to communicate with processing modules and provides the flexibility to adapt to changing standards. The interface enables an application to set the type of input and output data formats of a processing module and control when the processing module processes input data and generates output data. The processing modules enumerate its capabilities by category, by media type, or by both category and media type. Processing modules are registered by class ID, category, whether the application needs a key, the number and types of input data types, and the number and type of output data types to register.

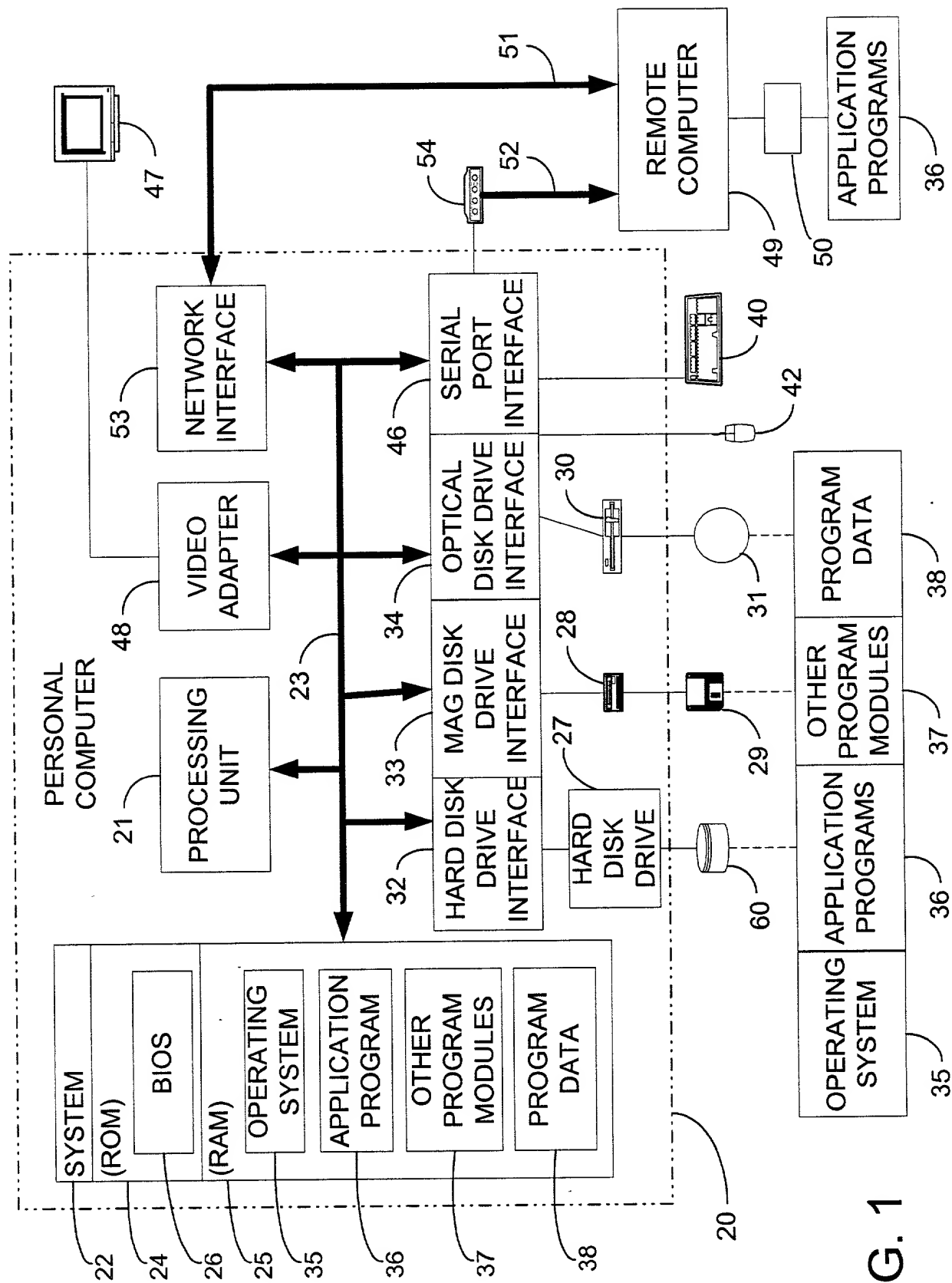


FIG. 1

FIG. 4
(PRIOR ART)

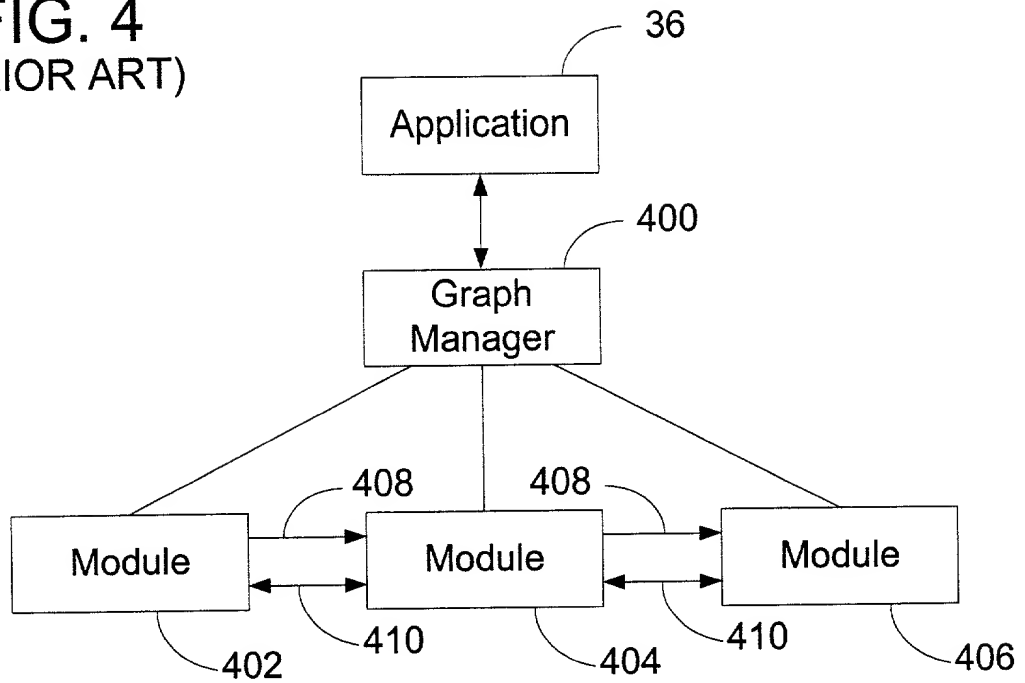


FIG. 2

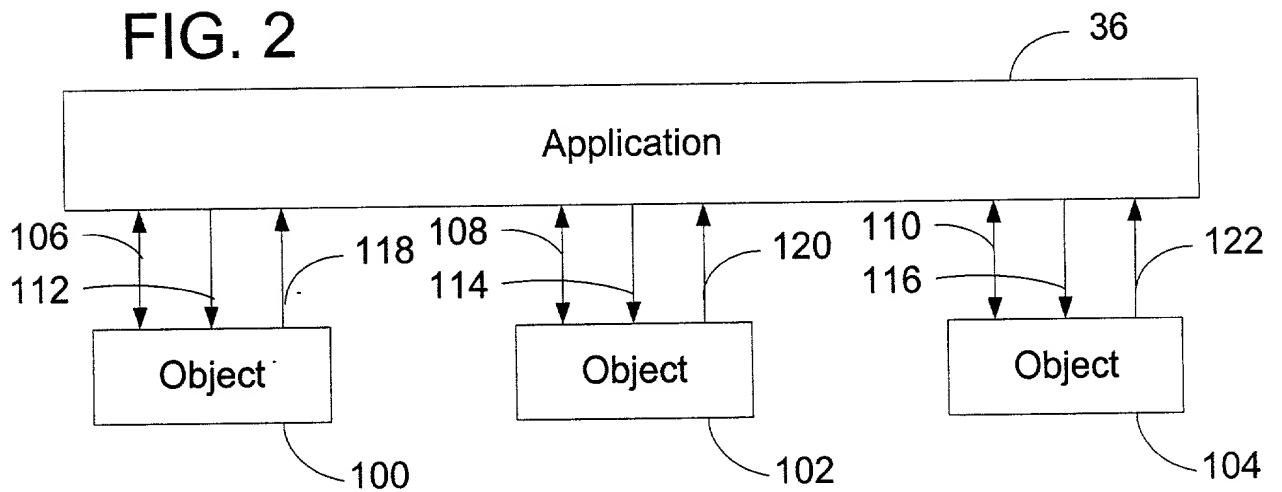
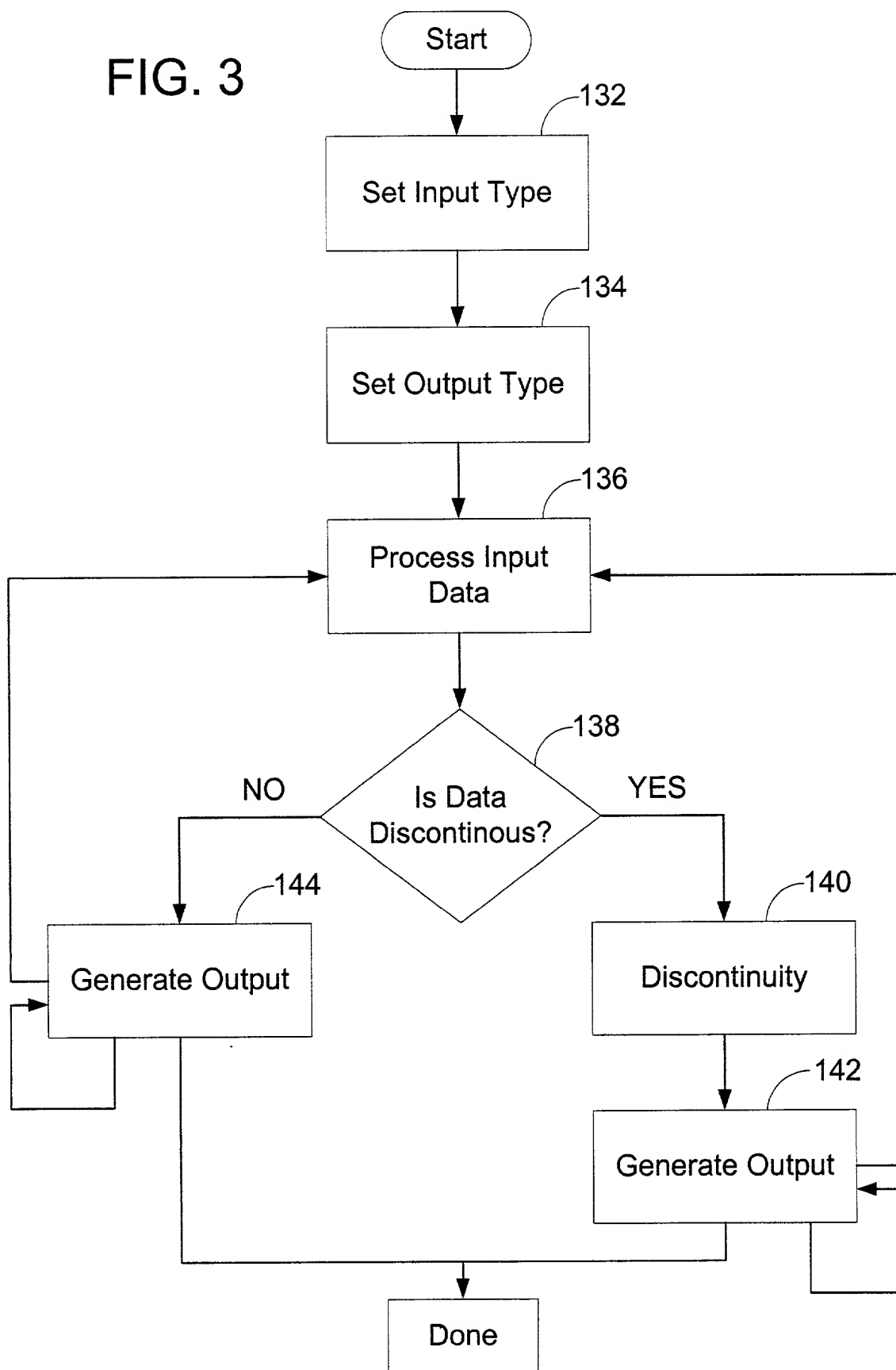


FIG. 3



The interface models the object as a set of input streams and a set of output streams each identified by a 0-based index.

The DMO_MEDIA_TYPE structure is used to identify all types of multimedia data.

IMediaObject methods and structures

Data buffers are wrapped inside an IMediaBuffer interface, a pointer to which is stored in each of the below structures. The IMediaBuffer interface contains a few simple methods to access the data pointer and the buffer's length:

```
interface IMediaBuffer : IUnknown
{
    HRESULT SetLength(
        [in] DWORD cbLength
    );
    HRESULT GetMaxLength(
        [out] DWORD *pcbMaxLength
    );
    HRESULT GetBufferAndLength(
        [out] BYTE **ppBuffer, // not filled if NULL
        [out] DWORD *pcbLength // not filled if NULL
    );
}
```

DMO_MEDIA_TYPE

```
{
    GUID    majortype;
    GUID    subtype;
    BOOL    bFixedSizeSamples;
    BOOL    bTemporalCompression;
    ULONG   lSampleSize;
    GUID    formattype;
    IUnknown *pUnk;
    ULONG   cbFormat;
    BYTE    *pbFormat;
}
```

This defines the media type structure. This structure exactly matches the DirectShow AM_MEDIA_TYPE structure and is given here for reference.

001E20"69252560

[illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible][illegible]

The types are enumerated in preference order with the most preferred type corresponding to a type index of 0.

If the call is successful the caller should free the output media type by calling **MoFreeMediaType()** when it's done.

For convenience of implementation it's possible that some types will be enumerated which will fail when used in **SetOutputType()**.

Parameters

dwInputStreamIndex	Zero based input stream index
dwTypeIndex	Zero based type index
pmt	Pointer to Media type to return

Return Values

S_OK	Success
S_FALSE	Type index out of range
DMO_E_INVALID_STREAM	Stream index out of range
E_OUTOFMEMORY	Could not allocate format block or some other memory failure
Failure code	Some other failure

HRESULT GetOutputType(DWORD dwOutputStreamIndex, DWORD dwTypeIndex, DMO_MEDIA_TYPE *pmt)

Get the **dwTypeIndex** type for the output stream **dwOutputStreamIndex**. The Media Type returned in **pmt** will be overwritten if the method is successful. The format block of the Media Type must be freed by calling **CoTaskMemFree()**.

The types are enumerated in preference order with the most preferred type corresponding to a type index of 0.

If the input type is not set for some input stream this call may fail or return a type with a NULL format block.

If the call is successful the caller should free the output media type by calling **MoFreeMediaType()** when it's done.

For convenience of implementation it's possible that some types will be enumerated which will fail when used in **SetOutputType()**.

Parameters

dwOutputStreamIndex	Zero based output stream index
dwTypeIndex	Zero based type index
pmt	Pointer to Media type to return

Return Values

S_OK	Success
S_FALSE	Type index out of range
DMO_E_INVALID_STREAM	Stream index out of range

00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

E_OUTOFMEMORY	Could not allocate format block or some other memory failure
Failure code	Some other failure

HRESULT SetInputType(DWORD dwInputStreamIndex, const DMO_MEDIA_TYPE *pmt, DWORD dwFlags)

Set a Media Type for an input stream. This call is processed in the context of the types currently set for other streams.

Parameters

dwInputStreamIndex	Zero based input stream index
pmt	Type to set
dwFlags	This must be a combination of the following flag values (or 0)
	DMO_SET_TYPEF_TEST_ONLY Just check if this type can be set, do not set it
	DMO_SET_TYPEF_CLEAR Clears the type so that no type is set for this stream

Return Values

S_OK	Type was set
S_FALSE	Type cannot be set
DMO_E_TYPE_NOT_ACCEPTED	Type is not acceptable
Failure code	Some other failure
E_INVALIDARG	Invalid argument

HRESULT SetOutputType(DWORD dwOutputStreamIndex, const DMO_MEDIA_TYPE *pmt, DWORD dwFlags)

Set a Media Type for an output stream. This call is processed in the context of the types currently set for other streams.

Parameters

dwOutputStreamIndex	Zero based output stream index
pmt	Type to set
dwFlags	This must be a combination of the following flag values (or 0)
	DMO_SET_TYPEF_TEST_ONLY Just check if this type can be set, do not set it
	DMO_SET_TYPEF_CLEAR Clears the type so that no type is set for this stream

Return Values

S_OK	Type can be set
-------------	-----------------

S_FALSE	Type cannot be set
DMO_E_TYPE_NOT_ACCEPTED	Type is not acceptable
Failure code	Some other failure

HRESULT GetInputCurrentType(DWORD dwInputStreamIndex, DMO_MEDIA_TYPE *pmt)

Get the Media Type for an input stream.

If the call is successful the caller should free the output media type by calling **MoFreeMediaType()** when it's done.

Parameters

dwInputStreamIndex	Zero based input stream index
pmt	Type returned here The type must be freed by calling MoFreeMediaType() if this call was successful

Return Values

S_OK	Type was returned
DMO_E_TYPE_NOT_SET	Type is not set
E_OUTOFMEMORY	Format block could not be allocated
Failure code	Some other failure

HRESULT GetOutputCurrentType(DWORD dwOutputStreamIndex, DMO_MEDIA_TYPE *pmt)

Get the Media Type for an output stream.

If the call is successful the caller should free the output media type by calling **MoFreeMediaType()** when it's done.

Parameters

dwOutputStreamIndex	Zero based output stream index
pmt	Type returned here The type must be freed by calling MoFreeMediaType() if this call was successful

Return Values

S_OK	Type was be set
DMO_E_TYPE_NOT_SET	Type is not set
E_OUTOFMEMORY	Format block could not be allocated
Failure code	Some other failure

HRESULT GetInputSizeInfo(DWORD dwInputStreamIndex, DWORD *pcbSize, DWORD *pcbMaxLookahead, DWORD *pdwAlignment)

Get buffer size and alignment requirements for a given input stream.

This method should be called after the types of all streams have been set using **SetInputType()** and **SetOutputType()**.

pcbMaxLookahead is only used for objects which hold on to multiple input buffers for lookahead. In that case the application must allow for enough buffers so that this amount can be retained by the object in order to generate output. For example, if the application decides on a fixed buffer size of **dwBufferSize** then it should be prepared to allocate up to at least:

$$(*pcbMaxLookahead + 2 * (dwBufferSize - 1)) / dwBufferSize$$

buffers of that size to avoid running out of buffers. This number may be reduced if there are alignment requirements in the data.

Parameters

dwInputStreamIndex	Zero based input stream index
pcbSize	Returns buffer size This is at least the minimum size required for processing
pcbMaxLookahead	Maximum size of data held by this object if it AddRefs multiple input buffers Or 0.
pdwAlignment	Returns buffer alignment. 1 means no alignment requirement.

Return Values

S_OK	Call successful
E_POINTER	NULL pointer passed in
Failure code	Other failure

HRESULT GetOutputSizeInfo(DWORD dwOutputStreamIndex, DWORD *pcbSize, DWORD *pdwAlignment)

Get buffer size and alignment requirements for a given output stream.

This method should be called after the types of all streams have been set using **SetInputType()** and **SetOutputType()**.

Parameters

dwOutputStreamIndex	Zero based output stream index
pcbSize	Returns buffer size
pdwAlignment	Returns buffer alignment. 1 means no alignment requirement.

Return Values

S_OK	Call successful
E_POINTER	NULL pointer passed in

001E20" 58262960

The Media Object should generate all output which can be generated from the data already received in calls to **ProcessInput()** on this stream before accepting more data on this stream.

If the **Discontinuity()** method has been called on all input streams for a Media Object and all output has been processed from all the output streams by calls to **ProcessOutput()** then the Media Object is in the equivalent state to the flushed state. In this state all buffers must be released and no more output can be generated until **ProcessInput()** is called again. Also in this state calling **Flush()** has no effect.

dwInputStreamIndex

0-based input stream index

S_OK

Call was successful

Flush internally buffered data and reset any streaming state. Media types and other parameters such as latency are not changed.

All streams should accept input after a **Flush()** call.

Return Values

Successful Failure

HRESULT AllocateStreamingResources()

Hint to allocate any resources necessary for processing. This method may not be called before the first call to **ProcessOutput()** and it is not required to support this method.

Parameters**Return Values**

S_OK	Successful. Return this if the call is not implemented.
Failure code	Some failure occurred

HRESULT FreeStreamingResources()

Hint to free any resources necessary for processing.

Parameters**Return Values**

S_OK	Successful. Return this if the call is not implemented.
Failure code	Some failure occurred

HRESULT GetInputStatus(DWORD dwInputStreamIndex, DWORD *pdwFlags)

Return input stream status.

Parameters

dwInputStreamIndex	0-based input stream index
pdwFlags	DMO_INPUT_STATUSF_ACCEPT_DATA This stream is ready to accept data

The setting of this flag can only change as the result of one of the following calls:

ProcessOutput()
Discontinuity()
ProcessInput()
Flush()

Return Values

001320 6626262

HRESULT ProcessInput(DWORD dwInputStreamIndex, IMediaBuffer *pBuffer, DWORD dwFlags, REFERENCE_TIME rtTimeStamp, REFERENCE_TIME rtTimeLength)

Deliver an input buffer for a stream. The Media Object should either process all the data inside this method or call **IMediaBuffer::AddRef()** to hold the buffer waiting for calls to **ProcessOutput()**. When the Media Object has generated all the output it can from this buffer it should call **IMediaBuffer::Release()** unless it needs the buffer for lookahead.

If the Media Object calls **IMediaBuffer::AddRef()** the application should not reuse a buffer until the matching **IMediaBuffer::Release()** is called.

Parameters

dwInputStreamIndex	Zero based input stream index
pBuffer	Buffer containing data Cannot be NULL
dwFlags	Must be a combination of the following flag values (or 0) DMO_INPUT_DATA_BUFFERF_TIME rtTimeStamp is valid DMO_INPUT_DATA_BUFFERF_TIMELENGTH rtTimeLength is valid DMO_INPUT_DATA_BUFFERF_SYNCPOINT Syncpoint at the beginning of the data
rtTimeStamp	Start timestamp in 100ns units
rtTimeLength	Length in 100ns units

Return Values

S_OK	Processed normally
S_FALSE	No output.
DMO_E_NOTACCEPTING	Data cannot be accepted until previous output has been processed by calling ProcessOutput()

HRESULT ProcessOutput(DWORD dwReserved, DWORD cOutputBufferCount, DMO_OUTPUT_DATA_BUFFER *pOutputDataBuffers, DWORD *pdwStatus)

Generate outputs from current input data. The status fields in the output data buffers are updated as a result of this call.

The **IMediaBuffer** pointers in the **DMO_OUTPUT_DATA_BUFFER** structures should not be held by **AddRef** after return from this call (ie their reference counts should be the same on exit as on entry).

Output buffer status fields are undefined if this call returns a failure code.

After calling **ProcessOutput()** the application should check all output streams for the **DMO_OUTPUT_DATA_BUFFERF_INCOMPLETE** flag. It is possible, for instance, when there are multiple output streams, for a stream which did not report **DMO_OUTPUT_DATA_BUFFERF_INCOMPLETE** previously to report it after a subsequent call to **ProcessOutput()**.

Parameters

dwFlags	DMO_PROCESS_OUTPUT_DISCARD_WHEN_NO_BUFFER If the pBuffer member of one of the output buffer structures is NULL discard output data.
cOutputBufferCount	Count of input buffers - this should be the same as the number of output streams.
pOutputDataBuffers	Array of output data buffers of size cOutputBufferCount.
pdwStatus	The Media Object should return 0 here.

Return Values

S_OK	Processing was successful
Failure code	Failure in processing

HRESULT GetInputMaxLatency(DWORD dwInputStreamIndex, REFERENCE_TIME *prtMaxLatency)

Returns the maximum latency in time between input on the stream and the corresponding output timestamps. Thus, for example, if input timestamped at time T generates output for time T-Delta then this value is the maximum possible value of Delta for the media types defined. This value does not take into account input buffer size.

Parameters

dwInputStreamIndex	0-based input stream index
prtMaxLatency	Latency

Return Values

E_NOTIMPL	Not implemented. Assume 0 latency
S_OK	OK
Failure code	Failure

HRESULT SetInputMaxLatency(DWORD dwInputStreamIndex, REFERENCE_TIME rtMaxLatency)

Sets the maximum latency in time between input on the stream and the corresponding output timestamps. Thus, for example, if input timestamped at time T generates output for time T-Delta then this bounds the maximum possible value of Delta for the media types defined. This value does not take into account input buffer size.

Parameters

dwInputStreamIndex	0-based input stream index
prtMaxLatency	Latency

Return Values

E_NOTIMPL	Not implemented. Latency cannot be set.
S_OK	OK
E_FAIL	Latency cannot be set

HRESULT Lock(LONG lLock)

Acquire a lock so that multiple operations can be performed while keeping the Media Object serialized.

Parameters

lLock	TRUE – lock FALSE - unlock
pvtMaxLatency	Latency

Return Values

S_OK	OK
E_FAIL	Cannot lock

Registration**DMO_PARTIAL_MEDIATYPE**

```
{
    GUID type;
    GUID subtype;
}
```

type	Major type for matching corresponding media types GUID_NULL means match any type
subtype	Subtype for matching corresponding media types GUID_NULL means match any subtype

HRESULT DMORegister(LPCWSTR szName, REFCLSID rclsidDMO, REFGUID rguidCategory, DWORD dwFlags, DWORD cInTypes, const DMO_PARTIAL_MEDIATYPE *pInTypes, DWORD cOutTypes, const DMO_PARTIAL_MEDIATYPE *pOutTypes)

Register a new object, its category and the media types it supports.

Parameters

szName	Registration name for this DMO. Names longer than 79 characters may be truncated.
rclsidDMO	Class ID the corresponding COM object for the DMO is registered under.
rguidCategory	Category of this object
dwFlags	This must be a combination of the following flag values (or 0). DMO_REGISTERF_IS_KEYED Object use is restricted to by key
cInTypes	Number of input types to register

plnTypes
cOutTypes
pOutTypes

Input types
 Number of output types to register
 Output types

Return Values

HRESULT DMOUnregister(REFCLSID rclsidDMO, REFGUID rguidCategory)

Unregister a media object from one or all categories.

Parameters

rclsidDMO
rguidCategory

Class ID of the DMO
 Remove from this category
 If this is GUID_NULL unregister this object
 from all categories

Return Values

**HRESULT DMOEnum(REFGUID rguidCategory, DWORD dwFlags, DWORD
 cInTypes, const DMO_PARTIAL_MEDIA_TYPE *pInTypes, DWORD cOutTypes,
 const DMO_PARTIAL_MEDIA_TYPE *pOutTypes)**

Enumerate Media Objects by category and/or by media type. GUID_NULL means match any GUID.

Parameters

rclsidDMO
rguidCategory
dwFlags

Class ID the corresponding COM object for the
 DMO is registered under.
 Category of this object
 This must be a combination of the following
 flag values (or 0).

cInTypes
plnTypes
cOutTypes
pOutTypes

DMO_REGISTERF_INCLUDE_KEYED
 Include keyed objects
 Number of input types to register
 Input types
 Number of output types to register
 Output types

Return Values

Media Type helpers

Use these functions to manipulate media types.

001E20168262960

Media types initialized with **MolnitMediaType** must be freed with **MoFreeMediaType**.
 Media types created with **MoCreateMediaType** must be freed with **MoDeleteMediaType**.
 Media types copied using **MoCopyMediaType** must be freed using **MoFreeMediaType**.
 Media types duplicated using **MoDuplicateMediaType** must be freed using **MoDeleteMediaType**.

HRESULT MolnitMediaType(DMO_MEDIA_TYPE *pmt, DWORD cbFormat)

Initialize a media type with a given size format block. **pmt** is assumed uninitialized on input and no attempt is made to free any media type previously in **pmt**.

Parameters

pmt	Where to initialize the media type
cbFormat	Size of format block to create

Return Values

E_OUTOFMEMORY

HRESULT MoFreeMediaType(DMO_MEDIA_TYPE *pmt)

Free a media type previously initialized by **MolnitMediaType**.
 On return the **pbFormat** field will be 0.

Parameters

pmt	Media type to free
------------	--------------------

Return Values

HRESULT MoCopyMediaType(DMO_MEDIA_TYPE *pmtDest, const DMO_MEDIA_TYPE *pmtSource)

Copy media types.

Parameters

pmtDest	Destination Media Type
pmtSource	Source Media Type

Return Values

E_OUTOFMEMORY	Could not allocate memory
---------------	---------------------------

HRESULT MoCreateMediaType(DMO_MEDIA_TYPE **ppmt, DWORD cbFormat)

Create a new media type structure.

Parameters

00000000000000000000000000000000

ppmt
cbFormat

Where to allocate the new media type
Size of format block

Return Values

E_OUTOFMEMORY

HRESULT MoDeleteMediaType(DMO_MEDIA_TYPE *pmt)

Delete a media type allocated by **MoCreateMediaType** or **MoDuplicateMediaType()** or returned by pointer from an API or interface method.

Parameters

pmt Media type to delete

Return Values

HRESULT MoDupliateMediaType(DMO_MEDIA_TYPE **ppmtDest, const DMO_MEDIA_TYPE *pmtSrc)

Duplicate a media type.

Parameters

ppmtDest New type
pmtSrc Source

Return Values

E_OUTOFMEMORY

001320"662560